

Transition Detection for Brain Computer Interface Classification

Ricardo Aler, Inés M. Galván, and José M. Valls

Computer Science Department, Carlos III University of Madrid
Avda. Universidad, 30, 28911 Leganés, Spain
{aler, igalvan, jvalls}@inf.uc3m.es
<http://www.evannai.uc3m.es>

Abstract. This paper deals with the classification of signals for brain-computer interfaces (BCI). We take advantage of the fact that thoughts last for a period, and therefore EEG samples run in sequences belonging to the same class (thought). Thus, the classification problem can be reformulated into two subproblems: detecting class transitions and determining the class for sequences of samples between transitions. The method detects transitions when the L1 norm between the power spectra at two different times is larger than a threshold. To tackle the second problem, samples are classified by taking into account a window of previous predictions. Two types of windows have been tested: a constant-size moving window and a variable-size growing window. In both cases, results are competitive with those obtained in the BCI III competition.

1 Introduction

The context of this paper is the brain-computer interface (BCI), a growing research field, that would allow users to control computers and other devices by means of brain signals [1], [10] [3], [8]. One of the main problems of BCI is to accurately decode individual brain signals. Machine Learning techniques are typically applied here, by training classifiers with the brain signals of the user that is going to use the interface [2]. For instance, a BCI can be trained to recognize three different classes corresponding to three different mental states: left-hand motor imagery, right-hand motor imagery, and object rotation imagery.

Noisy weak signals and high variability between same-user sessions¹ make the classification problem difficult, resulting in many on-line classification errors, frustrated users, and low transfer (speed) rates from the user to the computer. The transfer rate could be increased if three or more classes are used. However, multi-class classification problems are more difficult than two-class ones.

In this paper we aim to improve classification accuracy by taking advantage of a feature of BCIs: EEG samples (or instances) run in sequences belonging to the same class (i.e. the same thought), followed by a transition into a different class/thought. Typically, the BCI classification problem is tackled by trying to classify the EEG signal

¹ This means that the classifier learned during one session might not be accurate in the next session, even for the same user.

at every instant in time. However, given that classes run in sequences, the classification problem can be reformulated into two subproblems:

- Detecting class transitions
- Determining the class for sequences of instances between transitions.

The first problem can be approached in many different ways. In this paper we detect transitions by computing the L1 norm between the power spectra at two different times (instances) and signalling a class transition when the distance is larger than a threshold. Transition detection has also been used in [5] applying a proximity function based on three samples of the Power Spectrum Density.

By detecting transitions, the accuracy of the second problem (classification between transitions) can also be improved in two ways. First, if the class of the last sequence (before the transition) is known, that class can be discarded after the transition, hence becoming a simpler N-1 class classification problem. Also, it may be more accurate to classify an instance by taking into account several of the previous classifications. In this paper, two different types of windows are used. First, a ‘moving window’ with a fixed size. This kind of window only takes into account the n previous instances, where n has been fixed from the beginning. The second type of window is the ‘growing window’, with a variable size. When a transition is detected, the growing window size is initialized to zero and it keeps growing until the next transition.

The rest of the paper is organized as follows. Section 2 contains a description of the data supplied for the BCI III competition, that they are the data used in this work to validate and tested the proposed ideas. The method is described in Section 3. Section 4 shows the obtained results. Section 5 summarizes our results and draws some conclusions.

2 Description of EEG Data

In this paper we are going to use a high quality dataset acquired in the IDIAP Research Institute by Silvia Chiappa and José del R. Millán [9]. It was used in the BCI-III competition that took place in 2005.² This dataset contains data from 3 normal subjects during 4 non-feedback sessions. The subjects sat in a normal chair, relaxed arms resting on their legs. There are 3 tasks, so this is a three-class classification problem:

- Imagination of repetitive self-paced left hand movements
- Imagination of repetitive self-paced right hand movements
- Generation of words beginning with the same random letter

All 4 sessions of a given subject were acquired on the same day, each lasting 4 minutes with 5-10 minutes breaks in between them. The subject performed a given task for about 15 seconds and then switched randomly to another task at the operator’s request. EEG data is not splitted in trials since the subjects are continuously performing any of the mental tasks. Data was provided in two ways: raw EEG signals, and data with precomputed features. In this paper, we use the precomputed dataset.

² http://ida.fraunhofer.de/projects/bci/competition_iii/.

Features were precomputed as follows. The raw EEG potentials were first spatially filtered by means of a surface Laplacian. Then, every 62.5 ms (i.e., 16 times per second) the power spectral density (PSD) in the band 8-30 Hz was estimated over the last second of data with a frequency resolution of 2 Hz for the 8 centro-parietal channels C3, Cz, C4, CP1, CP2, P3, Pz, and P4. As a result, an EEG sample is a 96-dimensional vector (8 channels times 12 frequency components).

In summary, in this paper we are going to tackle a three-class classification problem with 96 input attributes, which define the PSD over 62.5 ms for 8 input channels. There are three subjects, with four datasets (sessions) for each one: three datasets for training and one dataset for testing.

3 Description of the Method

The method used here to improve BCI classification accuracy based on transitions and windows is based on two main ideas. First, the transition in the signal from a class to another class must be detected. This knowledge is used to discard the class assigned to the previous sequence of instances, just before the transition. Hence, the prediction problem of N classes is transformed into a prediction problem of $N - 1$ classes. Usually, in classification tasks the reduction of the number of classes helps to increase the performance of the classification algorithm.

On the other hand, the proposed method is inspired in the idea that in this problem, the class to predict remains fixed for a time period that we have called 'sequence of instances between transitions'. That means that, within a sequence, the class is not continuously changing. Hence, it makes sense to try to guess the class assigned to that sequence instead of using the classifier to predict each instance of the sequence independently of each other. The simplest way to assign a class to a sequence is to compute the majority class returned by the classifier on a small set of instances at the beginning of the sequence. However, if the classifier makes many mistakes at the beginning of the sequence, the whole sequence will be misclassified. In fact, in our first approach, we tried to assign a class to whole sequences based only on a few first instances, and very frequently, complete sequences were misclassified.

Therefore, we have decided to classify the i 'th instance in the sequence by considering a window of the n previous instances. We have tested two types of windows. In the first one, named 'moving window', the size n remains constant and therefore the window moves along with time. A heuristic method for computing the window size n will be explained at the end of the section. In the second case, named 'growing window', the size n keeps growing with time until the next transition. This means that in order to classify the i 'th instance, all instances since the last transition are considered. The moving window uses fewer previous instances to classify instance i , hence the classification accuracy might be lower. On the other hand, the growing window has more inertia: if mistakes are made by the user, the moving window will recover eventually, once the window has gone past the mistaken instances. But the growing window considers all instances and therefore it will require more additional correct instances in order to overcome the mistaken ones.

As we described in Section 2, we assume that the original data in the time domain has been transformed to the frequency domain using the Power Spectral Density (PSD).

Thus, we assume that we have a set of samples (or instances) and every sample contains all the components of the PSD for all channels at every instant. That is:

$$sample_i = \{PSD_{r,j}(i) \mid j = 1, \dots, NoC, r = 1, \dots, R\} \quad (1)$$

where $PSD_{r,j}(i)$ is the value of the r^{th} spectral component for the j^{th} channel, NoC is the number of channels, and R is the number of components in the PSD. For instance, if the PSD ranges from 8Hz to 30Hz and there is a component every 2Hz (a 2Hz resolution), then $R = 12$.

The classification method studied implies to detect transition and to decide the window size used to classify test instances. Hence, the method consists of different procedures which are described in detail in the next subsections and they are:

- The procedure to detect the transition
- The procedure to select the class once the transition is detected and one of the class is discarded
- The procedure to classify test instances with either a moving window or a growing window
- The mechanism to compute the window size of the moving window

3.1 Transition Detection

It is based on the observation that when a change of class occurs, a change in the frequency domain also occurs. The idea is to detect that transition by computing the L1 norm between PSDs at two consecutive samples. For every sample i , the distance $d_i = ||PSD(i) - PSD(i-1)||$ between the PSD of sample i and sample $i-1$ is calculated as:

$$d_i = \sum_{j=1}^{NoC} \left(\sum_{r=1}^R |PSD_{r,j}(i) - PSD_{r,j}(i-1)| \right) \quad (2)$$

Once the distance d_i is obtained, a threshold value U has to be set, such that if the distance is higher than this threshold, a transition is signaled. The value of U is crucial for successfully predict transitions and therefore, very important for the success of the method.

In order to determine that threshold, the following mechanism is applied. The training data set X is divided into two subsets, named $X_{nottransition}$ and $X_{transition}$. The first one, $X_{nottransition}$, contains the samples for which there is no transition; and, the second one $X_{transition}$ contains the transition samples. This separation can be made because samples are labeled in the training set (i.e. their class is known), and therefore a transition is just a change of class. After that, the distances d_i for samples in $X_{nottransition}$ are calculated and the maximum of these distances, named $MaxDis_{nottransition}$, is also obtained. The distances d_i for samples in subset $X_{transition}$ are also obtained and ranked from low to high. Obviously, a good threshold U must be larger than $MaxDis_{nottransition}$. We currently define U as the next d_i of set $X_{transition}$ that is larger than $MaxDis_{nottransition}$. This way of setting the threshold does not guarantee that all transitions will be detected. However, this method is simple and reasonable and later we will show that it performs correctly on the test data.

3.2 Choosing the Appropriate Classifier, Once the Transition Has Been Detected

Let us remember that when a transition is detected, the class of the previous sequence of instances is discarded (i.e. after the transition, the class must change). Therefore, after the transition, the system only needs a classifier that is able to classify samples (or instances) into $N - 1$ classes (discarding the previous class). As the system must be prepared for all classes, all $(N - 1)$ -class classifiers must be trained off-line using the training data, prior to the system going on-line. For instance, if we consider three classes (a , b and c), there will be three classifiers. The first classifier named K_{bc} , assumes that class a is discarded and will predict either class b or class c . The second classifier, named K_{ac} , will discard class b and classify instances in classes a or c . The last one K_{ab} , will consider only classes a and b .

Also, an N -class classifier must be learned, because at the beginning there is no previous sequence of samples, and therefore, no class can be discarded. In this work, we have used Support Vector Machines (SVM) because they obtain a very good performance[11], although any other machine learning algorithm could have been employed.³

The method is applied to predict the class in real time (on-line). The main idea is to use one of the classifiers for $N - 1$ classes, when a transition is detected, instead of using the general classifier (the classifier for N classes). Usually, the performance of classifiers with fewer classes is higher because if a class is removed, uncertainty about the class is also reduced.

The procedure to select the classifier in real-time is described next: when the prediction process begins, the N -class classifier is responsible for the prediction. At the same time, the distances d_i given by equation 2 are calculated. When a transition occurs (i.e. $d_i > U$), one of the N classifiers for $N - 1$ classes must be chosen. To choose the most appropriate classifier, the majority class predicted in the previous sequence of instances (i.e., the set of instances between the last two transitions) must be discarded and the $N - 1$ -class classifier corresponding to the remaining classes will be in charge of the predictions until the next transition is detected.

3.3 Procedure to Classify Test Instances with a Moving Window

As explained at the beginning of the section, we use a second idea to improve the classification accuracy: in order to classify $sample_i$, a window with the predictions of the selected classifier for the n previous samples will be used, instead of just using the prediction for $sample_i$. This is similar to determining if a biased coin is biased towards heads based only on the n previous few coin tosses. $sample_i$ will be classified as the majority class of instances within the window. It is a moving window because only the last n predictions just before $sample_i$ are taken into account. As the window moves, all the samples inside it are classified by the selected classifier. Figure 1 shows how it works. The only remaining issue is to estimate a "good" window size n . In this paper we have applied a heuristic method that will be explained later.

³ In fact, we use Weka's SMO implementation with standard parameters (a linear kernel and $C = 1$ [7]).

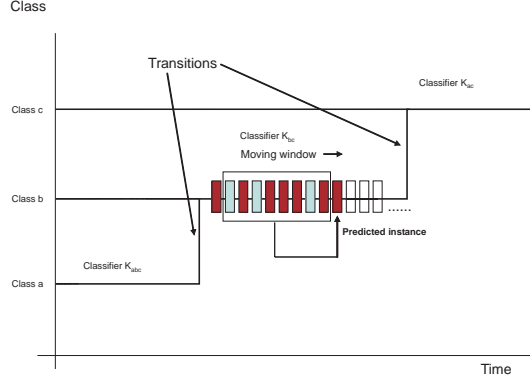


Fig. 1. Moving window to classify test instances

3.4 Procedure to Classify Test Instances with a Growing Window

We have also tried another approach named ‘growing window’ where the window size is not fixed, but it continues growing since the last transition. That is, all instances from the last transition to the current moment in time are included in the window. Initially the window is empty and continues growing until the next transition. Its main advantage is that it considers all possible instances. But it has an important drawback: it may take longer to recover from sequences of miss-classifications, because as the window always grows, it never forgets previous sequences of missclassified instances. This might be annoying for the user during on-line use, specially so after a long sequence of missclassifications due, for instance, to user fatigue.

3.5 Computing the Size of the Moving Window

Assigning the majority class of instances within the window to the next sample is reasonable, but mistakes would occur if the frequency of that class in the window is too close to 50%. This can be solved by establishing a safe confidence interval around the estimated frequency. For simplification purposes, let’s suppose there are only three classes ($N = 3$, classes a , b , and c), and as explained before, one of them will be discarded after the transition and one of the 3 2-class classifiers will be used for the current sequence until the next transition. Let’s suppose that class a is discarded and therefore classifier K_{bc} must be used for the current sequence (K_{bc} separates class b from class c). Although, the class of the current sequence is fixed until the next transition, the predictions of K_{bc} will make mistakes. In fact, just like coin tosses, classification errors follow a Binomial distribution (with success probability p). If the actual class of the current sequence is b , the distribution of mistakes of K_{bc} will follow a Binomial distribution with $p = TP_{K_{bc}}^b$, where $TP_{K_{bc}}^b$ is the True Positive rate for class b and classifier K_{bc} (i.e. the accuracy for class b obtained by classifier K_{bc}). On the other hand, if the actual class is c , $p = TP_{K_{bc}}^c$.

If the actual class is b , p can be estimated (\hat{p}) from a limited set of instances (we call it ‘the window’), and from standard statistical theory (and by assuming the Binomial distribution can be approximated by a Gaussian), it is known that \hat{p} belongs to a confidence interval around p with confidence α :

$$\hat{p} \in p \pm z_\alpha \sqrt{\frac{p(1-p)}{n}} \quad (3)$$

where n is the size of the window. From Eq 3, we can estimate the size of the window required:

$$n \geq z_\alpha^2 \frac{p(1-p)}{(p-0.5)^2} \quad (4)$$

When generating predictions, the actual class of the current sequence is not known, and therefore we have to assume the worst case, that happens when p is closest to 0.5. Therefore, $p = \min(TP_{K_{bc}}^b, TP_{K_{bc}}^c)$. To be in the safe side, for this paper, we have decided to make the window size independent of the classifier assigned to the current sequence. Therefore, if there are three classes $p = \min(TP_{K_{bc}}^b, TP_{K_{bc}}^c, TP_{K_{ab}}^a, TP_{K_{ab}}^b, TP_{K_{ac}}^a, TP_{K_{ac}}^c)$. A similar analysis could be done for more than three classes.

It is important to remark that Eq. 4 is only a heuristic approximation, since instances within a window are not independent of each other in the sense required by a Binomial distribution. The reason is that misclassifications are not uniformly distributed but happen in bursts (for example, if the user becomes tired for a period). Therefore, the probability of an instance being misclassified is not independent of previous instances. Yet, it seems heuristically appropriate that the window size is inversely proportional to $(p-0.5)^2$ (Eq. 4): the less accurate the classifier, the longer the window.

4 Results

The aim of this Section is to show the results of our method on the datasets described in Section 2. Let us remember that there were three subjects, and each one generated four sessions worth of data. The first three sessions are available for learning while session four is only for testing. All datasets are three-class classification problems with classes named 2, 3, and 7.

Our method computes all two-class SMO classifiers. SMO is the Weka implementation of a Support Vector Machine. Table 1 displays the results of all two-class classifiers (K_{23} , K_{27} , K_{37}) and the three-class classifier (K_{237}). The training has been made with sessions 1 and 2 and the testing with session 3. The three-class classifier accuracies can be used as a baseline to compare further results. In brackets we can observe the True Positive rate (TP) for each class. For instance, 74.7 is the True Positive rate (TP) for class 2 for the K_{23} two-class classifier (i.e. $TP_{K_{23}}^2$).

Section 3 gives the details for computing the thresholds for detecting transitions. These are: $U_1 = 0.563108963$, $U_2 = 0.58576$, $U_3 = 0.587790606$, for subjects 1, 2, and 3, respectively.

Our method uses the TP rate (class accuracy), obtained with session 3, for computing the moving window size, according to Eq 4. p will be set as the minimum of all TP

Table 1. Accuracy of two-class and three-class SMO classifiers for subjects 1, 2, and 3. Training with sessions 1 and 2, and testing with session 3.

SMO Classifier	Subject 1	Subject 2	Subject 3
K_{23}	79.0 (74.7/83.8)	71.9 (68.2/75.3)	52.3 (53.3/51.4)
K_{27}	82.4 (64.8/93.9)	74.3 (62.7/81.9)	57.9 (50.6/65.2)
K_{37}	83.0 (81.2/84.4)	76.8 (63.5/87.7)	60.4 (54.9/65.9)
K_{237}	73.8	62.0	40.9

rates. So we have $p_1 = 0.648$, $p_2 = 0.627$, and $p_3 = 0.506$ for subjects 1, 2, and 3, respectively. A confidence interval with $\alpha = 0.99$ will be used, therefore $z_\alpha = 2,5759$. Table 2 displays the windows sizes for every subject. It can be seen that the window size for subject 3 requires 46072 instances many more than available, so we apply the moving window idea only to subjects 1 and 2. This is due to the accuracy of classifier for subject 3 are very low, in particular the accuracy of K_{23} (see Table 1). We have also computed the window size for larger probabilities to check the performance of the method if smaller window sizes are used. For instance, we have also considered $p_1 = 0.80$ and $p_2 = 0.70$ (those values are approximately the accuracies of the two-class classifier in Table 1).

Table 2. Window size used for subjects 1, 2, and 3

	Probability	Window Size
Subject 1	0.648	78
Subject 2	0.627	92
Subject 3	0.506	46072
Subject 1	0.80	12
Subject 2	0.70	92

Finally, Table 3 shows the final results. The first row displays the competition results [4] on session four. [4] proposed an algorithm based on canonical variates transformation and distance based discriminant analysis combined with a mental tasks transitions detector. As required by the competition, the authors compute the class from 1 second segments and therefore no windows of samples are used. The second row, displays the best results from the competition using longer windows [6] (it reduces dimensionality of data by means of PCA and the classification algorithm is based on Linear Discrimination Analysis). No details are given for the size of the window of samples. The third row shows the results of the three-class classifier (sessions 1 and 2 were used for training and session 4, for testing).

The fourth row contains the results (on session four) of applying the transition detector only. In this case, once the transition is detected, the previous class is discarded

and the prediction is made using the two-class classifier chosen. These results are better than the three-class classifier. For subject 3, the performance of the method using the transition detection is very low because some of the two-class classifier for subject 3 have a very low accuracy (see K_{23} in Table 1). Let us remember that when a transition is detected, the previous class is discarded. The previous class is computed as the majority class of the previous sequence. If the classifier used in the previous sequence is very inaccurate, the majority class might not be the actual class. Hence, the wrong class would be discarded in the current sequence, and the wrong 2-class classifier would be selected. That would generate more mistakes that would be propagated into the next sequence, and so on. Given that all samples between transitions are used to compute the previous class and that the (N-1) classifiers are better than chance, the possibility of mistaking the previous class is very low. In fact, for the data used in this article, this situation has not occurred. But it is important to remark that preventing the propagation of the misclassification of the previous class is crucial for the success of the method and we intend to improve this aspect in the future.

The fifth row in Table 3 shows the results with the method described in Section 3. In this case, both ideas, the transition detection and the moving window size, are used. Results improve significantly if the moving window is used: classification accuracy raises from 74.8 to 94.8 and from 74.6 to 86.3 (subjects 1 and 2, respectively). It can also be seen that results are also improved if a smaller window size is used (sixth row: MW with small sample). The seventh row shows the classification rate for subject 1, 2 and, 3 for the growing window.

Comparing our method with the best competition result that used a window of samples (second row of Table 3), we can see that both the moving window and the growing window are competitive with respect to the first subject and improves the performance for the second subject. Regarding to the third subject, the growing window approach does not offer an improvement in accuracy. Also, the moving window cannot be applied to the third subject due to the large number of samples required for the window.

Table 3. Results for subjects 1, 2, and 3

	Subject 1	Subject 2	Subject 3
BCI competition (1 sec.)	79.60	70.31	56.02
BCI competition (long window)	95.98	79.49	67.43
3-class classifier	74.8	60.7	50.2
Transition detector	80.8	74.6	52.2
Moving window (ws = window size)	94.8 (ws=78)	86.3 (ws=92)	-
MW small sample	84.2 (ws=12)	82.5 (ws=35)	-
Growing window	96.9	88.0	61.7

It can also be seen that using windows improves accuracy significantly (second, fifth, sixth, and seventh rows versus the first one of Table 3).

5 Summary and Conclusions

It is known that in BCI classification problems, EEG samples run in sequences belonging to the same class (thought), and then followed by a transition into a different class. We present a method that takes this fact into consideration with the aim of improving the classification accuracy. The general classification problem is divided into two subproblems: detecting class transitions and determining the class between transitions. Class transitions are detected by computing the L1 norm between PSDs at two consecutive times; if the distance is larger than a certain threshold then a class transition is detected. Threshold values are automatically determined by the method. Once transitions can be detected, the second subproblem -determining the class between transitions- is considered. First, since the class before the transition is known, it can be discarded after the transition and therefore, a N -class problem becomes a $(N-1)$ -class problem, which is simpler. Second, in order to determine the class between transitions, a moving window is used to predict the class of each testing instance by taking into account the n last predictions. A heuristic estimation of the window size n is computed based on standard statistical theory. A growing window that takes into account all instances since the last transition has also been tested.

This method has been applied to a high quality dataset with precomputed features, resulting in a three-class classification problem with 96 input attributes. This dataset corresponds to three subjects, with four sessions for each one: three for training and one for testing. Several experiments have been done in order to validate the method and the obtained results show that just by applying the transition detector, the classification rates are better than when a 3-class classifier is used. When the moving and growing windows are used, the results are further improved. Results are competitive to the best ones obtained in the BCI competition: similar for subject 1, better for subject 2, and worse for subject 3. Predictions made by the growing window are more accurate than those of the moving window. However, the growing window might take longer to recover from bursts of misclassifications and therefore be less useful for on-line use. We have also shown that even if a smaller window size is used, the classification rates are still better than those that use only the transition detector and the 3-class classifier.

References

1. Curran, E.A., Stokes, M.J.: Learning to control brain activity: a review of the production and control of eeg components for driving brain computer interface (bci) systems. *Brain Cognition* 51 (2003)
2. Dornhege, G., Krauledat, M., Müller, K.-R., Blankertz, B.: General Signal Processing and Machine Learning Tools for BCI Analysis. In: *Toward Brain-Computer Interfacing*, pp. 207–234. MIT Press, Cambridge (2007)
3. Neuper, C., Pfurtscheller, G., Birbaumer, N.: In: *Motor Cortex in Voluntary Movements*, ch. 14, pp. 367–401. CRC Press, Boca Raton (2005)

4. Galan, F., Oliva, F., Guardia, J.: Bci competition iii. data set v: Algorithm description. In: Brain Computer Interfaces Competition III (2005)
5. Galán, F., Oliva, F., Guardia, J.: Using mental tasks transitions detection to improve spontaneous mental activity classification. *Medical and Biological Engineering and Computing* 45(6), 1741–0444 (2007)
6. Gan, J.Q., Tsui, L.C.S.: Bci competition iii. data set v: Algorithm description. In: Brain Computer Interfaces Competition III (2005)
7. Garner, S.: Weka: The waikato environment for knowledge analysis. In: Garner, S.R. (ed.) *WEKA: The waikato environment for knowledge analysis. Proc. of the New Zealand Computer Science Research Students Conference*, pp. 57–64 (1995)
8. Kubler, A., Muller, K.-R.: An Introduction to Brain-Computer Interfacing. In: *Toward Brain-Computer Interfacing*, pp. 1–26. MIT Press, Cambridge (2007)
9. del R. Millán, J.: On the need for on-line learning in brain-computer interfaces. In: *Proceedings of the International Joint Conference on Neural Networks*, Budapest, Hungary. IDIAP-RR 03-30 (July 2004)
10. Mourino, J., Millan, J., del, R., Renkens, F.: Noninvasive brain-actuated control of a mobile robot by human EEG. *IEEE Trans. Biomed. Eng.* 51 (2004)
11. Vapnik, V.: *Statistical Learning Theory*. John Wiley and Sons, Chichester (1998)